PRÁCTICA 4 - Uso del bus I2C y del sensor de temperatura

> Cuestión
>
> La dirección del sensor es 1000000 (es decir, 0x40 expresado en hexadecimal). Si queremos hacer una operación de lectura (bit R/W a 1), ¿cómo construiremos el segundo argumento de la llamada a i2c_master_write_byte() que haremos tras i2c_master_start() ?

La función `i2c_master_write_byte()` únicamente puede utilizarse cuando el dispositivo se encuentra configurado en modo **master** y su función es introducir en la cola de mensajes I2C un único byte de escritura (más información aquí). La función recibe tres parámetros, los cuales son relevantes de analizar:

- 1. La lista de comandos I2c donde se introducirán los datos.
- 2. Información que se introducirá en la lista especifica en la lista de comandos I2C.
- 3. Flag para activar la señal ACK.

Una vez visto esto, sabemos que el tercer parámetro es constante y el primero depende de la cola I2C que estemos utilizando, por lo que es más relevante es el segundo. Teniendo en cuenta que se trata de la primera escritura, necesitaremos establecer el modo de operación, de modo que el contenido de este será diferente dependiendo de si lo que queremos hacer es leer desde o escribir hacia el dispositivo, pero en ambos casos, dicho byte de información estará compuesto de la siguiente manera:

- Los primero 7 bits harán referencia a la dirección del esclavo al cual se dirige la escritura/lectura.
- El último bit dependerá del tipo de operación a realizar.

En el caso de Espressif, ya se encuentran definidas un par de macros para determinar si se produce una lectura o escritura, siendo estas **I2C_MASTER_READ** e **I2C_MASTER_WRITE** respectivamente. De esta manera, teniendo en cuenta que la dirección del dispositivo es de 7 bits, únicamente necesitaremos realizar un desplazamiento lateral de 1 bit sobre la variable que contiene dirección y operar mediante una puerta OR entre esta y la macro deseada.

En el siguiente cuadro tenemos un ejemplo sobre como sería la ejecución de dicha función para establecer una operación de lectura sobre el esclavo.

```
i2c_master_write_byte(cmd, (ESP_SLAVE_ADDR << 1) | I2C_MASTER_READ,
ACK_EN);
```

> Cuestión
>
> ¿Cuál es la diferencia entre 0xE3 y 0xF3? ¿Qué es clock stretching?

Para comprender las diferencias entre los comandos 0xE3 y 0xF3 necesitamos entender que el dispositivo **si7021** realiza dos tipos diferentes de instrucciones de lectura:

- La primera se realiza de forma periódica para llevar a cabo una medida relativa de la humedad y temperatura. Esta es utilizada por el comando **0xE0**, el cuál devuelve dicha medida relativa, por lo que no necesita llevar a cabo una nueva medición en el momento de ser invocado.

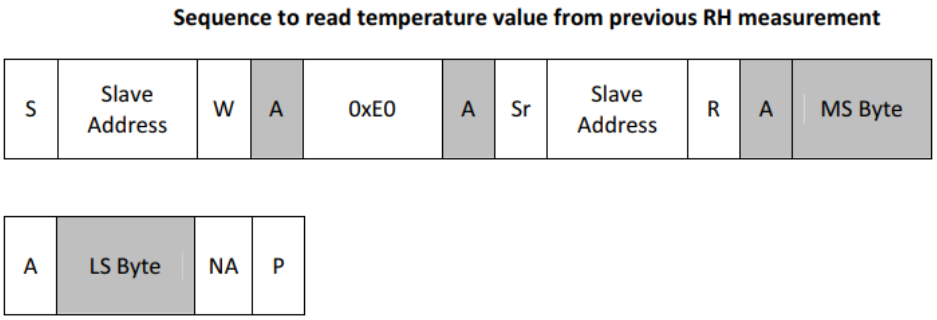
- La segunda se lleva a cabo en el mismo instante en el que es solicitado realizar la medición, sin embargo, esta forma de medición cuenta a su vez con dos maneras de llevarse a cabo, cada una utilizada mediante un comando diferente:

    - La medición se lleva a cabo indicando al dispositivo maestro cuando esta se ha realizado (Hold Mode). Esta se realiza mediante el comando **0xE3**.
    - La medición se lleva a cabo sin asentimiento, indicando al dispositivo maestro que se encuentra en proceso (No Hold mode) **0xF3**.

Por otra parte, cuando hablamos de **clock stretching** estamos haciendo referencia al uso del **Hold Mode** a la hora de llevar a cabo una medición.

> Cuestión
>
> Dichos comandos devuelven 2 bytes, que leeremos en dos variables diferentes. ¿Cómo obtenemos posteriormente nuestro número de 16 bits para calcular la temperatura?

Para responder correctamente a la pregunta, necesitaremos fijarnos la sección de la documentación sobre el periférico **si7021** donde nos muestra el esquema de envío de tramas referente a la realización de una lectura de temperatura. En la siguiente imagen podemos ver dicho esquema:



La primera parte de la cadena se corresponde con la conexión y el establecimiento del modo de lectura mediante la escritura en el dispositivo. Por el contrario si nos fijamos en los dos últimos mensajes enviados por el sistema esclavo, el cuál esta escrito en un color más claro, podemos ver que estos son llamados MS Byte (Most Significant) y LS Byte (Least Significant), los cuales corresponden con el bit de mayor y menor valor significativamente.

Una vez hemos obtenido ambos bytes de información, únicamente necesitaremos combinarlos en una variable de 16 bits y operar para obtener la temperatura en grados mediante la siguiente fórmula:

The results of the temperature measurement may be converted to temperature in degrees Celsius (°C) using the following expression:

$$\text{Temperature (°C)} = \frac{175.72 * \text{Temp\_Code}}{65536} - 46.85$$

Where:

Temperature (°C) is the measured temperature value in °C

Temp_Code is the 16-bit word returned by the Si7021

A temperature measurement will always return XXXXXX00 in the LSB field.

Ejercicio obligatorio - Uso de I2Ctools

> Ejercicio obligatorio
>
> Compila y prueba el ejemplo i2c_tools de la carpeta de ejemplos (examples/peripherals/i2c/i2c_tools). Conecta el sensor a los pines indicados por defecto (también a Vcc y a tierra) y ejecuta al comando i2cdetect. Prueba a los distintos comandos disponibles para tratar de leer información del sensor.

Para realizar el presente ejercicio, primero necesitaremos conectar el sensor **si7021** a los pines configurados por defecto dependiendo del SoC que estemos utilizando. Para esto podemos visualizar el fichero **README.md** del ejemplo, donde se nos indica cuales son las conexiones que deberemos llevar a cabo. En el siguiente cuadro podremos ver la configuración indicada:

```
**Note:** The following pin assignments are used by default, you can change
them with `i2cconfig` command at any time.

|                     | SDA    | SCL    | GND  | Other | VCC  |
| ------------------- | ------ | ------ | ---- | ----- | ---- |
| ESP32 I2C Master    | GPIO18 | GPIO19 | GND  | GND   | 3.3V |
| ESP32-S2 I2C Master | GPIO18 | GPIO19 | GND  | GND   | 3.3V |
| ESP32-S3 I2C Master | GPIO1  | GPIO2  | GND  | GND   | 3.3V |
| ESP32-C3 I2C Master | GPIO5  | GPIO6  | GND  | GND   | 3.3V |
| ESP32-C2 I2C Master | GPIO5  | GPIO6  | GND  | GND   | 3.3V |
| ESP32-H2 I2C Master | GPIO1  | GPIO2  | GND  | GND   | 3.3V |
| Sensor              | SDA    | SCL    | GND  | WAK   | VCC  |

**Note:** It is recommended to add external pull-up resistors for SDA/SCL
pins to make the communication more stable, though the driver will enable
internal pull-up resistors.
```

Una vez ejecutado el ejemplo, podremos ver un menú de opciones que nos muestras los diferentes comandos a poder utilizar. Lo primero será realizar una configuración del puerto I2C donde especifiquemos las conexiones que vamos a realizar mediante la instrucción `12cconfig`". En el siguiente cuadro tenemosS un ejemplo de configuración para nuestro SoC STM32:

```
i2c-tools> i2cconfig --port=0 --sda=18 --scl=19 --freq=100000
```

Posteriormente utilizaremos la instrucción `i2cdetect`, cuyo propósito es escanear el bus I2C en busca de posibles dispositivos conectados al mismo. Como resultado obtenemos la siguiente salida en forma de cuadro, la cual nos indica que detecta un total de dos dispositivos, uno situado en la dirección 0x00 y otro en la dirección 0x40. Este último se trata de nuestro sensor de temperatura y humedad.

```
i2c-tools> i2cdetect
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: 00 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

Una vez que ya conocemos la dirección del dispositivo, llevaremos a cabo una lectura de la temperatura medida por el mismo, la cual solicitaremos a través del acceso al registro 0xF3. Debemos tener en cuenta, que antes de poder realizar la lectura desde el dispositivo, necesitamos enviar un mensaje de escritura hacia el registro que queremos leer en cuestión. Esto quiere decir que deberemos realizar la siguientes secuencia:

1. Mensaje de escritura hacia el registro ha leer sin ningún parámetro en el campo Valor.
2. Mensaje de lectura del registro que queremos leer con una longitud lo suficientemente grande como para poder obtener toda la información necesaria (en nuestro caso 2).

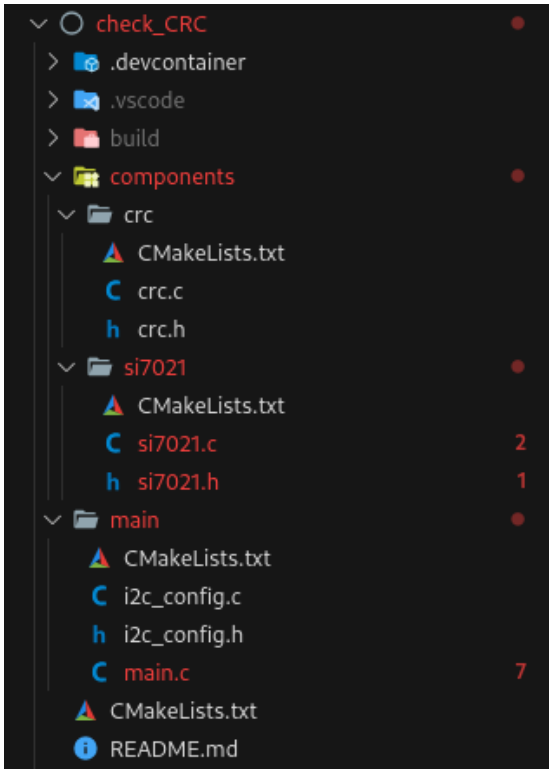
```
i2c-tools> i2cset -c 0x40 -r 0xF3
I (75432) cmd_i2ctools: Write OK
i2c-tools> i2cget -c 0x40 -l 2
0x65 0x44
```

En este punto ya hemos obtenido los datos de lectura de temperatura llevados a cabo por el sensor, por lo que únicamente necesitaríamos operar con los mismos para poder llevar a cabo el cálculo de dicho valor en grados, utilizando para ello la formula indicada anteriormente.

Verificación del CRC en el sensor si7021

> Ejercicio avanzado
>
> El sensor Si7021 permite el cálculo de un byte de checksum (CRC) para comprobar que no ha habido errores en el envío. Completa el código del componente para leer dicho byte y comprobar que no ha habido errores. Conviene leer la sección 5.1 y una librería para el cálculo de CRC como la ofrecida por BARR.

Para poder llevar a cabo el presente ejercicio vamos a utilizar tanto el componente **si7021** visto en prácticas anteriores como la librería de cálculo de CRC indicada en el enunciado. Por esto, el primer paso será crear un nuevo proyecto e integrar ambos componentes dentro del mismo, de manera que puedan ser visibles el uso desde el otro y a su vez utilizados desde la función `app_main()`. En la siguiente imagen podemos ver la estructura general del proyecto:



Una vez implementados todos los componentes, configuraremos el fichero **crc.h** para que en el momento de llevar a cabo el cálculo, este se realice de la misma manera que se desarrollo dentro del componente **si7021**, pudiendo ser así comparables. Debemos tener en cuenta la información que nos proporciona la guía del componente sobre el cálculo del crc:

- El cálculo se produce de forma polinómica mediante la fórmula **x8 + x5 + x4 + 1**.
- Se realiza una inicialización al valor **0x00**.

Teniendo esto en cuenta, la configuración llevada a cabo en dicho fichero será la siguiente:

```
#define CRC8


#if defined(CRC8)

typedef unsigned char  crc;

#define CRC_NAME            "CRC-8"
#define POLYNOMIAL          0x131
#define INITIAL_REMAINDER   0x0000
#define FINAL_XOR_VALUE     0x0000
#define REFLECT_DATA        FALSE
#define REFLECT_REMAINDER   FALSE
#define CHECK_VALUE         0x29B1
```

La verificación del crc obtenido se llevará a cabo en el componente **si7021**, donde se definirán nuevas funciones que tienen como objetivo llevar a cabo una lectura y realizar dicha comprobación. Además de esto, con el objetivo de poder mostrar los datos obtenidos y las operaciones con los mismos a nivel de bit, se han utilizado un par de funciones obtenidas desde el siguiente enlace, las cuales se ejecutarán mediante un modo *verbose*. En el siguiente cuadro podemos ver la operación con los datos dentro de dichas funciones en el fichero **si7021.h**:

```
//*****************************************************
// FUNCIONES AGREGADAS PARA REALIZAR EL CÁLCULO DE CRC
//*****************************************************

void print_byte_as_bits(char val);
void print_bits(char * ty, char * val, unsigned char * bytes, size_t
num_bytes);

esp_err_t readTemperature_withCRC(const i2c_port_t i2c_num, float
*temperature, bool *crcResult, bool verboseFlag);
esp_err_t _getSensorReading_withCRC(const i2c_port_t i2c_num, const uint8_t
command, float *output, bool *crcResult, bool verboseFlag, float (*fn)
(const uint16_t));
```

El pinto interesante del ejercicio es la comprobación del código crc devuelto por el periférico, lo cual se lleva a cabo dentro de la función `_getSensorReading_withCRC()`, una vez se han llevado a cabo todos los pasos de tratamiento de datos que se incluye en la versión de la misma aportada por al biblioteca, es decir, la que lleva a cabo la lectura sin la comprobación del crc, `_getSensorReading()`.

En el siguiente cuadro podemos ver el procedimiento llevado a cabo para la verificación del crc y la impresión de los datos por pantalla del modo **verbose**

```
    //Verificación del CRC
    unsigned char bytes2[] = {(unsigned char)buf[0], (unsigned char)
buf[1]};
    crcInit();
    unsigned char resultCRC = crcSlow(bytes2, sizeof(bytes2));

    if(verboseFlag){
        printf("Datos obtenidos de la lectura:\n");
        SHOW(unsigned char, buf[0]);
        SHOW(unsigned char, buf[1]);
        SHOW(unsigned char, buf[2]);
        printf("\n");

        printf("Datos pasados al calculador de CRC:\n");
        SHOW(unsigned char, bytes2[0]);
        SHOW(unsigned char, bytes2[1]);
        printf("\n");

        printf("Datos de salida:\n");
```

```
        SHOW(uint8_t, buf[2]);
        SHOW(unsigned char, resultCRC);
    }

    if(resultCRC == buf[2]){
        *crcResult = true;
    }else{
        *crcResult = false;
    }

    return ESP_OK;
}
```

Por último desarrollaremos la función `app_main()` para que lleva a cabo lecturas de temperatura periódicas mediante el uso de un hilo, implementando la definición de variables globales que nos permitan decidir el periodo de lectura y el uso del modo verbose. En el siguiente cuadro podemos el desarrollo de la lectura periódica de temperatura:

```
#define TIME_WAIT_TASK 2
#define VERBOSE_MODE 0

void taskShowTemperature();


void app_main(void)
{

    i2c_master_init();
    xTaskCreate(&taskShowTemperature, "taskShowTemperatura", 2048, NULL, 5,
NULL);
}

void taskShowTemperature()
{

    float temperature = 0;
    bool crcResult = (VERBOSE_MODE);

    while (1)
    {
        temperature = 0;
        crcResult = false;
        readTemperature_withCRC(I2C_MASTER_NUM, &temperature, &crcResult,
VERBOSE_MODE);
        printf("\nTemperatura %f  -  crcResult: %d\n\n\n", temperature,
crcResult);
        vTaskDelay(TIME_WAIT_TASK * 1000 / portTICK_PERIOD_MS);
    }
    vTaskDelete(NULL);
}
```

Para finalizar, vamos a llevar a cabo dos ejecuciones, una sin el uso del modo verbose y otra con el mismo activado. En los siguientes cuadros podemos ver ambas salidas respectivamente:

```
I (343) main_task: Calling app_main()
I (343) main_task: Returned from app_main()

Temperatura 25.780361  -  crcResult: true



Temperatura 25.812536  -  crcResult: true



Temperatura 25.801811  -  crcResult: true
```

```
I (343) main_task: Calling app_main()
I (343) main_task: Returned from app_main()
Datos obtenidos de la lectura:
(  unsigned char)           buf[0] = [ 01101001 ]
(  unsigned char)           buf[1] = [ 11100100 ]
(  unsigned char)           buf[2] = [ 10010011 ]

Datos pasados al calculador de CRC:
(  unsigned char)        bytes2[0] = [ 01101001 ]
(  unsigned char)        bytes2[1] = [ 11100100 ]

Datos de salida:
(        uint8_t)           buf[2] = [ 10010011 ]
(  unsigned char)        resultCRC = [ 10010011 ]

Temperatura 25.833986  -  crcResult: true
```